U-check: Model Checking and Parameter Synthesis under Uncertainty

U-check: Model Checking and Parameter Synthesis under Uncertainty

Luca Bortolussi[1 2 3⋆], Dimitrios Milios[4⋆⋆], and Guido Sanguinetti[4 5⋆⋆]

[1] Modelling and Simulation Group, University of Saarland, Germany
[2] Department of Mathematics and Geosciences, University of Trieste
[3] CNR/ISTI, Pisa, Italy
[4] School of Informatics, University of Edinburgh
[5] SynthSys, Centre for Synthetic and Systems Biology, University of Edinburgh

Abstract. Novel applications of formal modelling such as systems biology have highlighted the need to extend formal analysis techniques to domains with pervasive parametric uncertainty. Consequently, machine learning methods for parameter synthesis and uncertainty quantification are playing an increasingly significant role in quantitative formal modelling. In this paper, we introduce a toolbox for parameter synthesis and model checking in uncertain systems based on Gaussian Process emulation and optimisation. The toolbox implements in a user friendly way the techniques described in a series of recent papers at QEST and other primary venues, and it interfaces easily with widely used modelling languages such as PRISM and Bio-PEPA. We describe in detail the architecture and use of the software, demonstrating its application on a case study.

1 Introduction & Motivation

Tools and methodologies from formal analysis are increasingly playing a central role in science and engineering. Recent years have seen a veritable explosion in the number of novel application domains for quantitative analysis of dynamical systems, from systems biology, to smart cities, to epidemiology. A common feature of these novel application domains is the presence of uncertainty: while expert opinion may inform modellers about the presence of specific interactions (the *structure* of the model), it seldom is sufficient to quantify precisely the kinetic parameters underpinning the system dynamics. Extending formal analysis methods to handle models with parametric uncertainty is therefore rapidly becoming a major area of development in formal modelling.

Unsurprisingly, the trend towards modelling uncertain systems has led to a convergence between ideas from machine learning and formal modelling. While most efforts are focussing on the problem of identifying parameter values that

⋆ Work partially supported by EU-FET project QUANTICOL (nr. 600708) and by FRA-UniTS.
⋆⋆ Work supported by European Research Council under grant MLCS 306999.

may match particular specifications in terms of observations or global properties (*parameter synthesis*, [1,12,4,8,10]), more recent efforts aim at characterising and exploiting the dependence of system properties on the parametrisation, and embedding the concept of uncertainty in formal modelling languages [7,9,15]. Despite the considerable interest such approaches are generating, user-friendly implementations of machine learning methodologies for formal analysis are currently lacking.

In this paper, we present U-check, a toolkit for formal analysis of models with parametric uncertainties based on *Gaussian Processes* (GPs), a flexible class of prior distributions over functions underpinning many Bayesian regression and optimisation algorithms [21]. GPs are at the core of several novel developments in formal analysis [8,7,2,9,3,18]; here we focus on three particular tasks: estimating the parametric dependence of the truth probability of a linear temporal logic formula; synthesising parameters from logical constraints on trajectories, and identifying parameters that maximise the robustness (quantitative satisfaction score [13,2]) of a formula. Our tools are based on Java and interface with popular formal modelling programming languages such as PRISM [17] and Bio-PEPA [11]; we also offer support for hybrid models specified in the SimHyA modelling language (for stochastic hybrid systems) [5]. U-check is available to download at `https://github.com/dmilios/U-check`.

The problems tackled by U-check. U-check is a tool to perform model checking, parameter estimation, and parameter synthesis for uncertain stochastic models. We consider a parametric family of stochastic processes $\mathcal{M}_{\boldsymbol{\theta}}$, indexed by parameters $\boldsymbol{\theta}$ in a bounded subset $\mathcal{D} \subseteq \mathbb{R}^m$, usually a hyperrectangle. The parametric dependence is introduced to reflect the impossibility of removing uncertainty from model specification. We will refer to the pair $(\mathcal{M}_{\boldsymbol{\theta}}, \mathcal{D})$ as un *uncertain stochastic model*. Given a model $\mathcal{M}_{\boldsymbol{\theta}}$, we are often interested in understanding some features of its global behaviour, which can be specified as a set of formal properties, for instance using a linear temporal logic formalism as MiTL or STL [20,19]. U-check solves one of the following problems:

- **Smoothed Model Checking.** Given an uncertain stochastic model $(\mathcal{M}_{\boldsymbol{\theta}}, \mathcal{D})$, and a linear property φ, smoothed model checking provides a statistical estimate of the satisfaction probability of the formula as a function of the model parameters (*satisfaction function*), $p_{\varphi}(\boldsymbol{\theta})$; the tool also returns pointwise confidence bounds. The estimate is obtained in a Bayesian framework combining simulation to generate trajectories, a monitoring routine to very φ on the so obtained trajectories, and Gaussian Process-based statistical inference; details are provided in [7].
- **Parameter estimation from qualitative observations.** This algorithm takes as input an uncertain stochastic model $(\mathcal{M}_{\boldsymbol{\theta}}, \mathcal{D})$, n linear time properties $\varphi_1, \ldots, \varphi_n$, and N observations of the joint satisfaction value of such properties. Then, using an active learning optimisation algorithm based on Gaussian Processes [22], it computes the maximum likelihood (or the maxi-

mum a-posteriori) estimate $\hat{\boldsymbol{\theta}}$ of parameters $\boldsymbol{\theta}$ that best explain the observed dataset [8].
- **Robust parameter synthesis.** Given an uncertain stochastic model $(\mathcal{M}_{\boldsymbol{\theta}}, \mathcal{D})$ and a property φ, the algorithm identifies the parameters set $\boldsymbol{\theta}^*$ that maximises the expected robustness (satisfaction score) of φ, again exploiting a Gaussian Process-based optimisation algorithm [2].

2 Background material

U-check takes as input an uncertain stochastic model $(\mathcal{M}_{\boldsymbol{\theta}}, \mathcal{D})$ specified as a stochastic model $\mathcal{M}$ and of a range of values for (a subset of) its parameters. The primary focus of U-check is on Continuous-Time Markov Chain models [14], that are simulated by the standard stochastic simulation algorithm [16].

2.1 Population CTMC

A Continuous time Markov Chain (CTMC) $\mathcal{M}$ is a Markovian (i.e. memoryless) stochastic process defined on a finite or countable state space S and evolving in continuous time [14]. We will specifically consider population models of interacting agents [6], which can be easily represented by

- a vector of population variables $\boldsymbol{X} = (X_1, \ldots, X_n)$, counting the number of entities of each kind, and taking values in $S \subseteq \mathbb{N}^n$;
- a finite set of reaction rules, describing how the system state can change. Each rule η is a tuple $\eta = (\boldsymbol{r}_\eta, \boldsymbol{s}_\eta, f_\eta)$. $\boldsymbol{r}_\eta$ (respectively $\boldsymbol{s}_\eta$) is a vector encoding how many agents are consumed (respectively produced) in the reaction, so that $\boldsymbol{v}_\eta = \boldsymbol{s}_\eta - \boldsymbol{r}_\eta$ gives the net change of agents due to the reaction. $f_\eta = f_\eta(\boldsymbol{X}, \boldsymbol{\theta})$ is the rate function, associating to each reaction the rate of an exponential distribution, depending on the global state of the model and on a d dimensional vector of *model parameters*, $\boldsymbol{\theta}$. Reaction rules are easily visualised in the chemical reaction style, as

$$r_1 X_1 + \ldots r_n X_n \xrightarrow{f(\boldsymbol{X},\boldsymbol{\theta})} s_1 X_1 + \ldots s_n X_n$$

2.2 Property Specification

In this work, properties systems are expressed as properties of their trajectories via *Metric Interval Temporal Logic* (MiTL) [20]. Formally, the syntax of a MiTL formula φ is given by the following grammar:

$$\varphi ::= \mathtt{tt} \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathbf{U}_{[T_1,T_2]} \varphi_2 \tag{1}$$

where $\mathtt{tt}$ is the true formula, conjunction and negation are the standard boolean connectives, and there is only one temporal modality, the time-bounded until $\mathbf{U}_{[T_1,T_2]}$. Further connectives can be easily derived from the terms of the grammar above. For example, temporal modalities like time-bounded eventually and always can be defined as: $\mathbf{F}_{[T_1,T_2]}\varphi \equiv \mathtt{tt}\mathbf{U}_{[T_1,T_2]}\varphi$ and $\mathbf{G}_{[T_1,T_2]}\varphi \equiv \neg\mathbf{F}_{[T_1,T_2]}\neg\varphi$.

Given a system with n population variables, a trajectory will be a real-valued function $\mathbf{x}(t)$, $\mathbf{x} : [0, T] \to \mathbb{R}^n$. An atomic proposition μ transforms a function $\mathbf{x}(t)$, to a boolean signal $\mathbf{s}_\mu(t) = \mu(\mathbf{x}(t))$, where $\mathbf{s} : [0, T] \to \{\mathtt{tt}, \mathtt{ff}\}$. The truth of a formula φ with respect to a trajectory $\mathbf{x}$ at time t is given by the standard satisfiability relation $\boldsymbol{x}, t \models \varphi$. For instance, the rule for the temporal modality states that $\boldsymbol{x}, t \models \varphi_1 \mathbf{U}_{[T_1,T_2]} \varphi_2$ if and only if $\exists t_1 \in [t + T_1, t + T_2]$ such that $\boldsymbol{x}, t_1 \models \varphi_2$ and $\forall t_0 \in [t, t_1]$, $\boldsymbol{x}, t_0 \models \varphi_1$, while $\boldsymbol{x}, t \models \mu$ if and only if $\mathbf{s}_\mu(t) = \mathtt{tt}$.

A CTMC $\mathcal{M}_\theta$ is characterised by a distribution of random trajectories. In this context, a MiTL formula φ can be associated with the probability $Pr(\boldsymbol{x}, 0 \models \varphi | \mathcal{M}_\theta)$, which is the probability that the formula is satisfied at time zero by a trajectory $\boldsymbol{x}$ sampled from $\mathcal{M}_\theta$[6].

We also offer evaluation of MiTL formulae under quantitative semantics, which returns a measure of robustness for a given trajectory [2]. The quantitative specification function ρ returns a value $\rho(\varphi, \mathbf{x}, t) \in \mathbb{R} \cup \{-\infty, +\infty\}$ that quantifies the robustness degree of φ by the trajectory $\mathbf{x}$ at time t subject to perturbations. In the stochastic setting, the robustness will be a real-valued random variable R_φ which captures the distribution of robustness degrees over the trajectory space.

2.3 Statistical methodologies

We provide here a very brief intuition about the statistical methodologies employed by U-check. A full discussion is provided in the cited papers [7,8] and is beyond the scope of this tool paper.

The fundamental idea behind U-check is that the (intractable) functional dependence of a formula's truth probability on the model parameters can be abstracted through the use of statistical methods. We leverage the fact that the satisfaction probability of a temporal logic formula over an uncertain CTMC is a smooth function of the parameters, which was proved in [7]. Smoothness has important practical repercussions: knowing the value of a smooth function at a point $\mathbf{x}$ is informative about the value of the function in a neighbourhood of the point $\mathbf{x}$ through the Lipschitz property implied by smoothness. Intuitively, U-check exploits this transfer of information between neighbouring points to devise effective algorithms to explore (and optimise) the parametric dependence of truth probabilities.

More formally, the starting point for U-check is to obtain estimates of a truth probability at a set of initial points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ via a standard simulation-based statistical model checking algorithm. These values are treated as *noisy observations* of the unknown function (the truth probability). We then proceed in a Bayesian framework, place a Gaussian Process (GP) prior over the function values at *all* parameter values and combine this with the observations to obtain a *posterior* estimate. GPs are infinite dimensional generalisations of Gaussian distributions [21]; crucially, they can easily encode smoothness and inherit many

[6] We assume implicitly that T is sufficiently large so that the truth of φ at time 0 can always be established from $\boldsymbol{x}$.

favourable computational properties from the finite dimensional case. GPs therefore enable us to construct a statistical surrogate of the satisfaction function; this is the central idea behind all of the techniques implemented in U-check. The specific algorithms used for optimisation and smoothed model checking are different, and are described in detail in the main references of the paper.

3 Software Architecture

One of the main requirements of U-check is to be a multi-platform tool, which can also be easily incorporated as a library in other software projects. For this reason, the entire system has been implemented in Java. U-check depends on separate projects that have been developed independently, namely the PRISM project, the Bio-PEPA project, and the SimHyA project, which offer functionality to load models written in the respective languages. Other external libraries that our tool depends on are jBLAS, which is a linear algebra library for Java, and Apache Commons Math, which we use for local optimisation routines.

The software components of U-check, along with their dependencies are summarised in Figure 1. The main functionality of the tool is implemented by two main components: Learning From Formulae and Smoothed Model Checking. The former depends on the GP optimisation component which implements the non-convex optimisation algorithms required. Both components depend on the Gaussian Process framework and the Model Checking component, which offers routines to perform statistical model checking for CTMCs. Finally, the U-check CLI component offers the functionality of all of the components involved in a common command line interface.

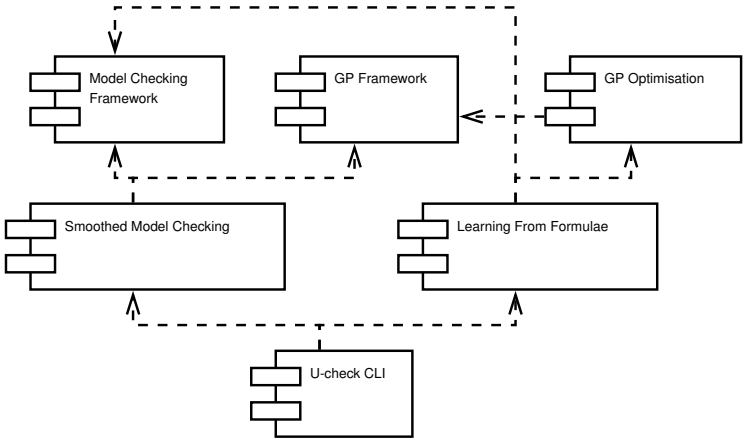

Fig. 1. Component Diagram for U-check

3.1 Gaussian Processes Framework

This module is responsible for the GP regression tasks, on which the main methodologies that we cover rely. **Regular GP Regression** assumes that the data are noisy observations of the latent function, which is then analytically approximated by a series of linear algebra calculations. We offer two ways of handling the observation noise either as a constant defined by the user, or using an automatically calibrated heteroskedastic noise model. **Probit Regression** is used in the case of smoothed model checking, where the objective is to emulate satisfaction probability as a function of the parameters. The output of the emulated function has to be strictly in the interval $[0, 1]$, hence regular regression is no longer appropriate and has to be combined with a probit transformation.

Both regression approaches require the specification of a covariance function; we use the *Radial Basis Function* (RBF) kernel due to its theoretical properties [7]. For the RBF kernel, there is an isometric version labelled as `rbfiso`, and a version that supports a different lengthscale parameter for each dimension of the input space. The latter is labelled as `rbfard` and it can be combined with hyperparameter optimisation to achieve *automatic relevance determination*. To automatically determine the hyperparameters of the kernel, we offer two alternatives: a heuristic that relies on the range of the training observations, or a local hyperparameter optimisation using the marginal likelihood of the observed data. For this local optimisation task (using the heuristic as default initialisation), we use the optimisation toolkit of Apache Commons.

3.2 GP Optimisation Framework

This module constitutes an implementation of the GP optimisation algorithm, which is described in [8], as a generic framework for non-convex optimisation of noisy objective functions. The module depends on the GP framework, as it utilises regular GP regression to emulate a given objective function.

GP Optimisation is initialised with a random grid of points, that is used as training set in a GP model. The GP posterior is calculated over a random set of test points; for each test point we calculate the estimated value of the objective function and its associated variance. The emulated value, along with the corresponding variance, serves as an indicator whether there is a potential maximum nearby. The GP regression model is used to direct the search towards areas of the search-space that have not been explored adequately. The strategy is that the test point that maximises an upper quantile of the GP posterior is selected to be added to the training set. This step is repeated for a number of iterations, and therefore the training set is progressively updated with new potential maxima, until a certain convergence criterion is satisfied. For more details see [8].

3.3 Model Checking Framework

This component is responsible for parsing and evaluating MiTL properties. The successful parsing of a MiTL formula will result in an abstract syntax tree which

can be evaluated over a specified trajectory. A trajectory can be either a random sample from a CTMC model, or a solution to a system of ordinary differential equations (ODEs). The formulae can be evaluated either in terms of the standard boolean semantics, which is used for smoothed model checking and parameter inference from qualitative data, or the quantitative semantics, which is used for robust parameter synthesis.

The module also involves stochastic simulation routines for CTMCs, which are used by default to evaluate the satisfaction probabilities of MiTL formulae. The simulation capabilities can be overridden, if the model checking module is used as a library. Regarding the modelling component, we have defined an interface that accepts different implementations; in the current version we offer implementations based on PRISM, Bio-PEPA and SimHyA.

3.4 Smoothed Model Checking

This component combines the Gaussian process framework with the model checking capabilities, in order to construct analytic approximations to the satisfaction probability as a function of the model parameters. The structure is outlined in the UML class diagram of Figure 2. The specification of the model and the properties to be verified is responsibility of `MitlModelChecker` class. The `ModelInterface` specifies the method signatures for loading models, setting the model parameters, and generating trajectories. The model checking framework is agnostic of the modelling and simulation details; it is therefore easily expandable to different kinds of implementations. The form of the trajectories to be generated is part of the interface (the `Trajectory` class), so that these are compatible with the `MiTL` class, which implements an abstract syntax tree of MiTL expressions.

The `SmoothedModelChecker` class is responsible for the main functionality of the module, which makes use of `MitlModelChecker` and `GPEP` to construct an analytic approximation of the satisfaction function. The respective class, i.e. `AnalyticApproximation`, is essentially a trained probit regression GP model. The `performSmoothedModelChecking` method uses this analytic result to estimate the satisfaction probability for a number of points in the parameter space. The ranges of the parameters to be explored are specified by the `Parameter` class, while the `SmmcOptions` class controls the configuration options of a smoothed model checking experiment, further discussed in Section 4.

3.5 Learning From Formulae

This module depends on the model checking and the GP optimisation framework. The module is responsible for the application of the GP optimisation algorithm to parameter synthesis, which is achieved by appropriate objective functions.

The structure of the module is outlined in Figure 3. The `GPOptimisation` class implements the GP optimisation algorithm; it relies on `RegressionGP`,

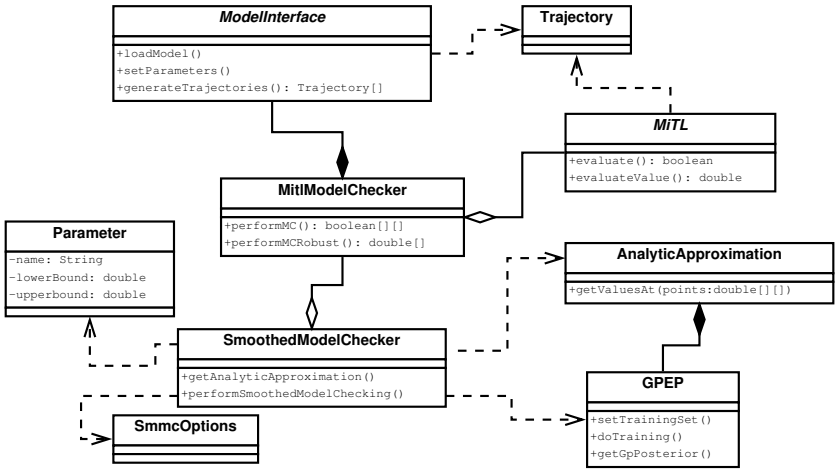

Fig. 2. UML class diagram for the Smoothed Model Checking component

while its options are controlled by `GpoOptions`. As already discussed in Section 3.4, `MitlModelChecker` is responsible for the specification of models, properties and the simulation algorithms. The `LearnFromFormulae` class performs the actual parameter synthesis. It has to be initialised with an object of type `MitlModelChecker`, a set of parameters and their corresponding prior distributions. The latter are represented by the abstract `Prior` class, whose implementations offer different options, including uniform, exponential, Gaussian and gamma distributions. Other options are specified by `LFFOptions`, which involve options regarding the simulation algorithms used, and the entire set of options in `GpoOptions`. The `GpoResult` class contains the result of the optimisation process; that involves the optimal solution found, along with a covariance matrix that captures the uncertainty of the approximated optimum. In a Bayesian setting, this is interpreted as a Gaussian approximation of the posterior distribution of the parameters.

3.6 U-check CLI

This module is the implementation of the command-line tool for model checking and parameter synthesis under uncertainty. It offers a common API to provide the functionality of both Learning From Formulae and Smoothed Model Checking components. It is responsible for linking the other components of U-check with the PRISM, SimHyA and Bio-PEPA libraries, providing implementations to the specified interfaces. It also provides functionality of reading the required experiment options from a configuration file, whose structure is outlined in the next section.

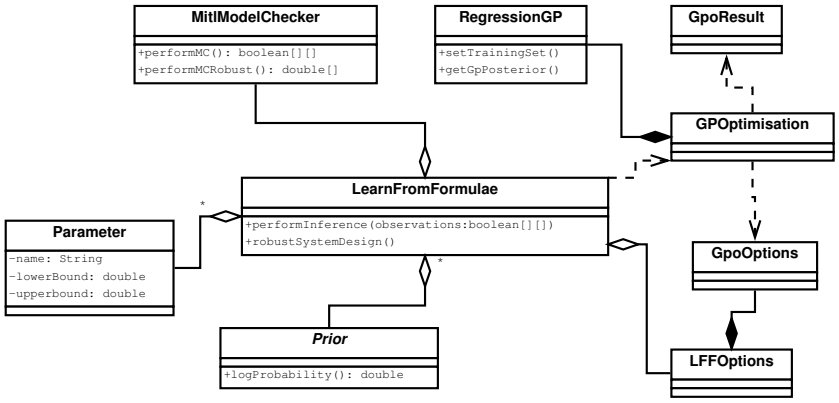

Fig. 3. UML class diagram for the Learning From Formulae component

4 Configuration Options

We describe the practical usage of U-check on an example in the next section. Here we summarise the configuration options of the program, avoiding excessive details for readability; a comprehensive description of the options is given in the user manual associated to the code release. The command line interface of our tool dictates that the program is provided with a configuration file, which lists a number of options in the form of assignments as follows:

```
OPTION = VALUE
```

where `VALUE` can be either a number, a truth value, or a string, depending on the nature of the option. Only a few configuration options are necessary to produce results, while the rest are optional and their corresponding default value will be used if no explicit assignment is made.

4.1 Experiment Configuration

The following options control the main setting of each experiment, which involves the definition of the model, the properties and the mode of operation.

- `modelFile`: A file that contains the specification of a population CTMC described either in PRISM, Bio-PEPA or the SimHyA language.
- `propertiesFile`: A file that contains the specification of one or more MiTL properties.
- `observationsFile`: A file that contains a $n \times m$ matrix, whose rows correspond to n independent observations of the system in question. A row is a single observation and contains m values (0 or 1), one for each one of the MiTL formulae specified in the properties file.
- `mode`: It can be either `inference`, `robust` or `smoothedmc`.

Parameter uncertainty is expressed as a range of possible parameter values, which is provided by the user in addition to model specification. Each parameter has to be associated with an interval, which is specified with an assignment of the form:

```
parameter NAME = [A, B]
```

Optionally, each parameter can be associated with a prior distribution using an assignment of the form:

```
prior NAME = uniform(A, B) | exponential(MU)
           | gamma(ALPHA, BETA) | gaussian(MU, S2)
```

Only independent univariate priors are supported. If no prior is explicitly declared, then a uniform prior will be assumed. The prior information is only utilised in the parameter inference scenario.

In order for an experiment to progress, `modelFile`, `propertiesFile`, `mode` and at least one parameter are required to be specified in the configuration file. In the case of parameter inference from qualitative data, `observationsFile` is also required to be specified.

4.2 Simulation Options

The options in this category control the parameters of the simulation process. The most important of these are `endTime`, which sets the time up to which the system will be simulated, and `runs` which controls the number of the independent simulation runs per parameter value.

It is also possible to use simulation engines other than stochastic simulation via the `simulator` option. This can take one of three values: `ssa` for Gillespie's stochastic simulation algorithm, `odes` for mean-field approximation, and `hybrid`. The `odes` and the `hybrid` simulation are based on the implementation of SimHyA [5], and they support SimHyA models only. Moreover, the mean-field approximation by ODEs can only be used for robust parameter synthesis.

4.3 GP Options

The options in this category determine the properties of the GP regression models used in the tool. The size of the training set for the GP is defined by the `initialObservations` option. For the parameter synthesis operations, it affects the initialisation of the GP optimisation algorithm. For smoothed model checking, it controls the initial evaluations of the satisfaction function via statistical model checking. In general, increasing this value is expected to increase the approximation quality for the GP regression model. However, an excessively large value for this parameter implies that the entire process degenerates to naïve parameter space exploration via statistical model checking. It is recommended to begin with a relatively small value (for example 100, which is the default value), and progressively increase until the results are estimated with adequate confidence.

The `numberOfTestPoints` option controls the size of the test set for the GP process. In case of parameter synthesis, that is the set of points where the GP posterior is estimated at each step, in order to find a new potential global maximum. Increasing the size of this set will increase the chances of discovering a new potential maximum. For smoothed model checking, the test set contains the points at which we explore the satisfaction function. Alternatively, the test set can be specified via the `testPointsFile` option: a set of parameter values is directly specified in a csv file that contains a $n \times d$ matrix, each row of which is a d-dimensional point in the parameter space.

The `kernel` option defines the kind of the covariance function used: either `rbfiso` for isometric RBF kernel, or `rbfard` for RBF kernel that supports *automatic relevance determination*. Other options control the hyperparameters of the kernel; by default these are internally optimised via a local optimisation process.

GP Optimisation Options For the modes of operation that involve parameter optimisation, there are a number of options available that control the convergence properties of the GP optimisation algorithm. The optimisation process is considered to have converged, if a certain number of added points with no significant improvement is reached, defined by the `maxAddedPointsNoImprovement` option. An improvement is considered significant if: $f_{n+1} > f_n * \alpha$, where α is the improvement factor, also set by the user. Convergence is alternatively assumed if a number of failed attempts to find a new local optimum is reached, which is set via `maxFailedAttempts`.

5 Case Study

We shall demonstrate the use of U-check on a rumour-spreading model, whose PRISM specification is outlined in Figure 4. There are three modules that correspond to the population variables of a PCTMC; these are spreaders, ignorants and blockers. A spreader and an ignorant may interact via the `spreading` action, which means that the ignorant is converted to a spreader. If two spreaders interact via the `stop_spreading1` action, then one of them will become blocker and therefore stop spreading the rumour. Finally, the `stop_spreading2` action dictates that a blocker may convert a spreader to blocker.

All rates of this model follow the law of mass action, with kinetic constants `k_s` and `k_r`. We shall measure the probability that the rumour has not reached the entire population, assuming that there has been some initial outbreak. We consider the following MiTL property, which states that there will be still ignorants between time 3 and 5, while the population of spreaders has climbed above 50% of the population before time 1. Note that the nested globally term indicates that the spreader population has to remain above 50 for at least 0.02 time units.

$$\varphi_1 = \mathbf{G}_{[3,5]}\, \mathit{ignorants} > 0 \ \wedge\ \mathbf{F}_{[0,1]}(\mathbf{G}_{[0,0.02]}\, \mathit{spreaders} > 50) \tag{2}$$

```
ctmc
const double k_s=0.05;
const double k_r=0.02;

module spreaders
 spreaders : [0..100] init 10;
 [spreading] true -> spreaders : (spreaders'=spreaders+1);
 [stop_spreading1]  true -> spreaders * (spreaders - 1)
                            : (spreaders'=spreaders-1);
 [stop_spreading2]  true -> spreaders
                            : (spreaders'=spreaders-1);
endmodule

module ignorants
 ignorants : [0..100] init 100-10;
 [spreading] true -> ignorants : (ignorants'=ignorants-1);
endmodule

module blockers
 blockers : [0..100] init 0;
 [stop_spreading1] true -> 1 : (blockers'=blockers+1);
 [stop_spreading2] true -> blockers : (blockers'=blockers+1);
endmodule

module base_rates
 [spreading]       true -> k_s : true;
 [stop_spreading1] true -> k_r : true;
 [stop_spreading2] true -> k_r : true;
endmodule
```

Fig. 4. PRISM model specification for a rumour-spreading system

An example of property specification as used in U-check is shown in Figure 5. A property file contains a list of constant declarations, followed by one or more modal expressions.

5.1 Smoothed Model Checking

We demonstrate the application of smoothed model checking, considering uncertain parameters `k_s` and `k_r`. The performance of the approach in terms of approximation quality and efficiency compared to naïve parameter exploration has been analysed in [7]. Given a model file specification `rumour.sm`, and a property file `rumour.mtl` that contains the formula in (2), the contents of the configuration file to perform smoothed model checking will be the following:

```
modelFile = rumour.sm
propertiesFile = rumour.mtl
mode = smoothedmc
```

```
// constant declarations
const int threshold = 3.75

// MiTL properties
G[3,5] ignorants > 0 & F[0,1] (G<=0.02 spreaders>=threshold)
```

Fig. 5. An example of a properties file

```
parameter k_s = [0.0001, 2]
parameter k_r = [0.0001, 0.5]
endTime = 5
runs = 10
initialObservtions = 100
numberOfTestPoints = 625
```

According to the `initialObservtions` option, U-check will evaluate the satisfaction probability on a grid of 100 regularly distributed parameters values between 0.0001 and 2 for `k_s` and between 0.0001 and 0.5 for `k_r` correspondingly. The `numberOfTestPoints` option means that the GP posterior will be evaluated on a grid of 625 points.

The results are written in a text file named `MODEL.csv`, where `MODEL` is the name of the input model. The output file contains the grid of input points along with the associated predictions and confidence intervals. The program also produces a script file named `load_MODEL.m`, that allow easy manipulation of the results under either matlab or octave. The tool also supports limited plotting capabilities via the gnuplot program. If only one parameter is explored, the satisfaction function is plotted along with the confidence bounds obtained by the GP. In the two dimensional case, the satisfaction function is depicted as a 2-D map. Out-of-the-box visualisation of higher-dimensional data is not currently supported. An example of plots produced automatically by U-check can be seen in Figure 6.

5.2 Robust Parameter Synthesis

We next use U-check to maximise the robustness of the property in (2). In this case, the `initialObservations` and the `numberOfTestPoints` options control the GP training and test sets in the optimisation context.

```
modelFile = rumour.sm
propertiesFile = rumour.mtl
mode = robust
parameter k_s = [0.0001, 2]
parameter k_r = [0.0001, 0.5]
endTime = 5
runs = 100
initialObservations = 100
numberOfTestPoints = 50
```

We quote the part of the program output that contains information regarding the solution obtained. For `k_s`, the most robust value is 0.341, while for `k_r` we

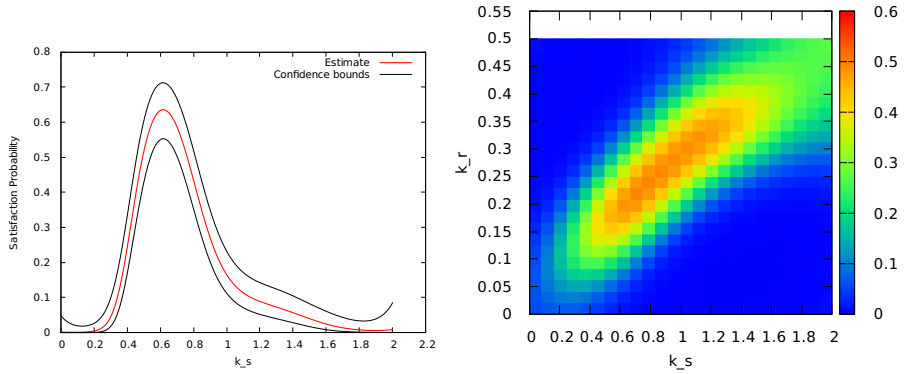

Fig. 6. Emulated satisfaction probability of φ_1 as function of the parameters. Left: We vary `k_s` only, while `k_r` is fixed to 0.2. Right: We vary both `k_s` and `k_r`.

have optimal value equal to 0.107. Note however that the standard deviations calculated are significantly large compared to the estimates. This is an indication that the optimum obtained is unstable. In this case, this is due to the fact that the robustness of the property in question is not very sensitive to the parameters.

```
# Gaussian Process Optimisation --- Results
Solution:      [0.3410609996706949, 0.10736369126944924]
Standard Dev: [3.6248813872572243, 0.7830513406533174]
Covariance matrix:
[13.13976507168386, 2.838061284295024;
 2.838061284295024, 0.6131694020989578]
```

5.3 Inference from Qualitative Data

Finally, we present an example of performing parameter inference from qualitative data. We consider two additional MiTL properties:

$$\varphi_2 = \mathbf{F}_{[0,1]}\, blockers \geq spreaders$$
$$\varphi_3 = \mathbf{G}_{[1.4,2]}\, spreaders \leq 25$$

The φ_2 property states that at some point before time 1 the population blockers surpasses the spreader population, while φ_3 states that the spreaders are always fewer that 25 between time 1.4 and 2. We have considered the rumour-spreading model with parameters `k_s`= 0.05 and `k_r`= 0.02, and we have produced a synthetic file of observations named `rumour.dat`, by performing model checking on random trajectories of the original model. U-check then requires the following input:

```
modelFile = rumour.sm
propertiesFile = rumour.mtl
observationsFile = rumour.dat
mode = inference
```

```
parameter k_s = [0.0001, 2]
parameter k_r = [0.0001, 0.5]
prior k_s = exponential(0.5)
prior k_r = exponential(0.1)
endTime = 5
runs = 100
```

As for robust parameter synthesis, the results of the optimisation process involve the optimal value obtained for each parameter and the corresponding standard deviation. Note that we have a particularly good fit for both parameters; the estimate for `k_s` is 0.050, and for `k_r` is 0.0243, which are very close to the original values, with a low standard deviation.

```
# Gaussian Process Optimisation --- Results
Solution:     [0.05098993039106938, 0.02430176520400998]
Standard Dev: [0.007751880832319672, 0.0019051431370011513]
Covariance matrix:
[6.009165643848513E-5, 0.0;
 0.0, 3.629570372462587E-6]
```

6 Conclusions

Uncertainty is increasingly recognised as an unavoidable companion in many applications of formal methods. This has motivated an increasing cross-fertilisation of ideas between machine learning and quantitative formal modelling. In this paper we describe U-check, a novel tool which implements a number of Gaussian Process based methods for formal analysis of uncertain stochastic processes. Our aim is to offer a set of tools that can be used by modellers without an in-depth knowledge of statistical machine learning. To our knowledge, U-check is the first such tool available; to further facilitate adoption of the tool, U-check can take as input models formulated in widely used modelling languages such as PRISM and Bio-PEPA.

The principal conceptual innovation of the methods implemented in U-check is the smoothness of the satisfaction function for a MiTL formula as a function of the model parameters. This enables us to transfer information across neighbouring observations, yielding potentially very significant computational savings: while we cannot comprehensively review results here, Smoothed Model Checking was shown in [7] to yield computational savings of an order of magnitude on non-trivial systems biology models. Similarly, the smoothness of the satisfaction function enables us to deploy a provably convergent algorithm for (robust) parameter synthesis, with both theoretical guarantees and computational advantages, as shown in non-trivial case studies in [8,2].

While in our work we primarily focus on examples from biology, uncertain stochastic processes are the norm in many other areas of application of formal methods, from smart cities to cyber-physical systems. Future work will explore increasing support for hybrid systems, as well as supporting other formal analysis methodologies such as reachability computations [9] and property synthesis [3].

References

1. A. Andreychenko, L. Mikeev, D. Spieler, and V. Wolf. Approximate maximum likelihood estimation for stochastic chemical kinetics. *EURASIP Journal on Bioinformatics and Systems Biology*, 2012(1):1–14, 2012.
2. E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. On the robustness of temporal properties for stochastic models. In *Proc. of HSB*, volume 125, pages 3–19, 2013.
3. E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proc. of FORMATS*, pages 23–37, 2014.
4. E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka. Model repair for probabilistic systems. In *Proc. of TACAS*, pages 326–340. 2011.
5. L. Bortolussi, V. Galpin, and J. Hillston. *Hybrid performance modelling of opportunistic networks*, pages 106–121. Number 85 in EPTCS. 2012.
6. L. Bortolussi, J. Hillston, D. Latella, and M. Massink. Continuous approximation of collective systems behaviour: a tutorial. *Performance Evaluation*, 70:317–349, 2013.
7. L. Bortolussi, D. Milios, and G. Sanguinetti. Smoothed model checking for uncertain continuous time Markov chains. *CoRR*, abs/1402.1450, 2014.
8. L. Bortolussi and G. Sanguinetti. Learning and designing stochastic processes from logical constraints. In *Proc. of QEST*, pages 89–105. 2013.
9. L. Bortolussi and G. Sanguinetti. A statistical approach for computing reachability of non-linear and stochastic dynamical systems. In *Proc. of QEST*, pages 41–56, 2014.
10. M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *Proc. of CMSB*, pages 86–98, 2014.
11. F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009.
12. R. Donaldson and D. Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *Proc. of CMSB*, pages 269–287, 2008.
13. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Proc. of FORMATS*, pages 92–106. 2010.
14. R. Durrett. *Essentials of Stochastic Processes*. Springer, 2012.
15. A. Georgoulas, J. Hillston, D. Milios, and G. Sanguinetti. Probabilistic programming process algebra. In *Proc. of QEST*, pages 249–264, 2014.
16. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.
17. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV*, pages 585–591, 2011.
18. A. Legay and S. Sedwards. Statistical abstraction boosts design and test efficiency of evolving critical systems. In *Proc. of ISOLA*, pages 4–25, 2014.
19. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proc. of FORMATS*, pages 152–166, 2004.
20. J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *Proc. of FORMATS*, pages 1–13, 2008.
21. C. Rasmussen and C. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.
22. N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Information-theoretic regret bounds for Gaussian process optimisation in the bandit setting. *IEEE Trans. Inf. Th.*, 58(5):3250–3265, 2012.